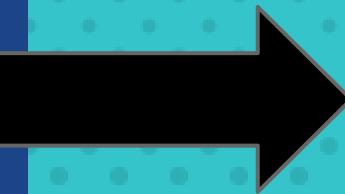


# **POLIMORFISMO**

**PARTE 1**

# Programação Orientada a Objetos



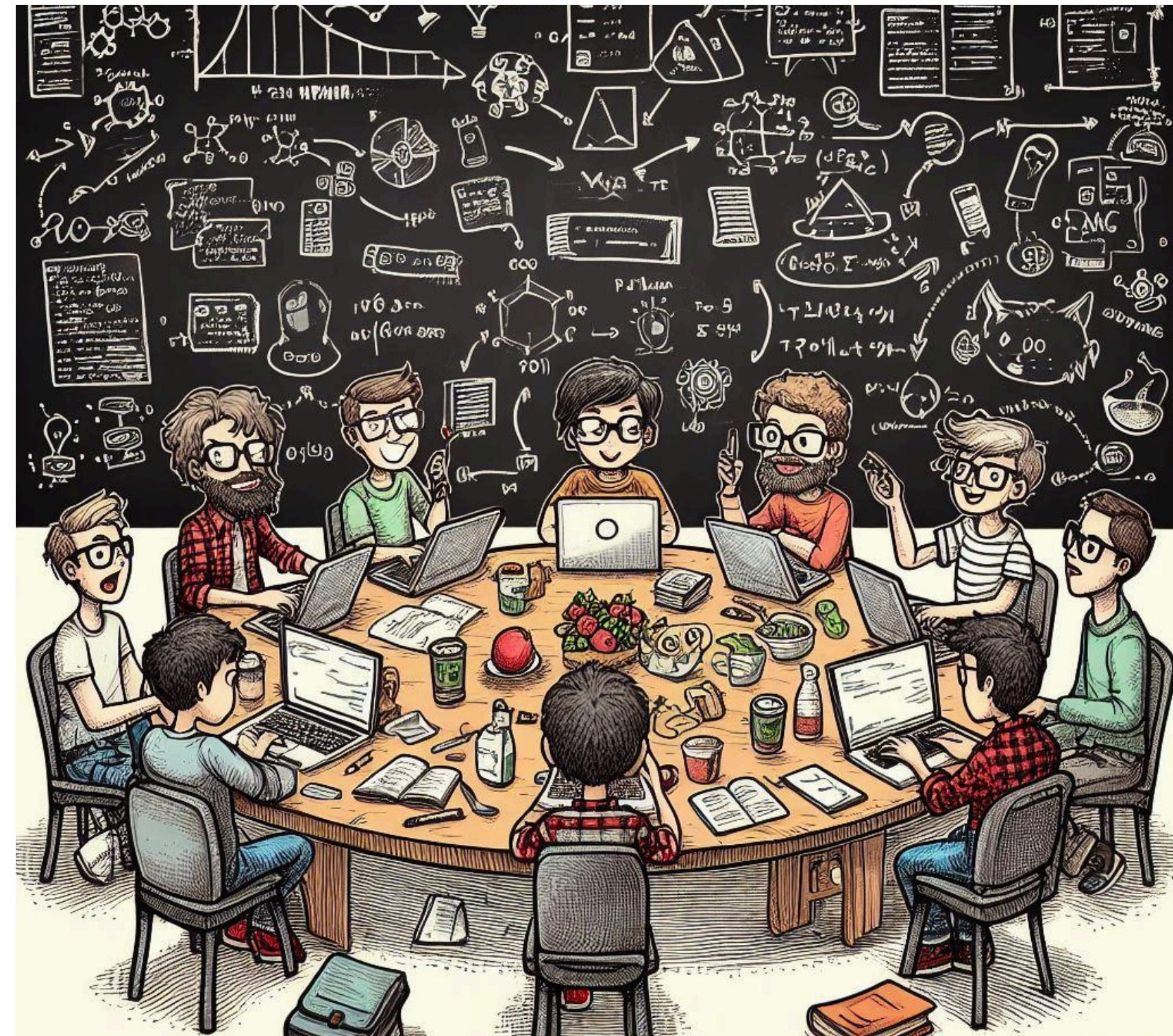
Abstração

Encapsulamento	Construtores	Objetos	Métodos	Atributos	Classes	
Teste de Software	Modificadores de acesso	Métodos de acesso	Métodos modificadores	Getters e Setters	Métodos estáticos	Herança
Polimorfismo	Reutilização	Sobrescrita de métodos	Associação	Extends	Super e subclasse	
	Classes abstratas	instanceof e as operator	métodos abstratos	interfaces	Generics	END

# **ORGANIZAÇÃO**

Atividade  
timebox para ser  
realizada em  
grupo

Escolher um  
líder/representante



# **OBJETIVO**

Exercitar herança e  
identificar pontos de  
melhoria.

# **MISSÃO**

**Crap Games** quer desenvolver um novo jogo, e solicitou a você que desenvolva a estrutura de classes dos personagens do game chamado "Walking in crap".

**Utilize** todas as boas práticas de programação conhecidas.

**Os personagens** do game serão:  
guerreiro, mago, padre

# TAREFA 0 - PERSONAGEM

Esta servirá como superclasse de classes especializadas

**Características:** nome, força, habilidade mental, poder de ataque, esquiva, resistência, vida atual e vida máxima.

**Ações:**  
atacar

Entrada (parâmetro): Um personagem

Retorno: Sem retorno

Comportamento: Desconhecido

**contra-atacar**

Entrada (parâmetro): Um personagem

Retorno: Sem retorno

Comportamento: Desconhecido

**aprimorar habilidade principal**

Entrada (parâmetro): Sem entrada

Retorno: Sem retorno

Comportamento: Desconhecido

**regenerar vida**

Entrada (parâmetro): Sem entrada

Retorno: Sem retorno

Comportamento: Desconhecido

Atenção!

getters e setters podem e devem ser criados conforme necessidade para qualquer classe do projeto.

?? minutos

# TAREFA 0 - PERSONAGEM

● ● ●

```
1  export class Personagem {  
2    constructor(  
3      protected _nome: string,  
4      protected _forca: number,  
5      protected _habilidadeMental: number,  
6      protected _podeDeAtaque: number,  
7      protected _esquiva: number,  
8      protected _resistencia: number,  
9      protected _vidaAtual: number,  
10     protected _vidaMaxima: number  
11   ) {}  
12  
13   public atacar(personagem: Personagem): void {  
14     console.log("Um comportamento desconhecido");  
15   }  
16  
17   public contraAtacar(personagem: Personagem): void {  
18     console.log("Comportamento desconhecido");  
19   }  
20  
21   public aprimorarHabilidadePrincipal(): void {  
22     console.log("Comportamento desconhecido");  
23   }  
24  
25   public regenerarVida(): void {  
26     console.log("Comportamento desconhecido");  
27   }  
28 }  
29
```

# TAREFA 1 - GUERREIRO

?? minutos

## Características:

**nome** - O nome de um guerreiro deve possuir o sufixo "Warrior". Ex.: Quando cadastro um guerreiro com o nome "Ragnar", será registrado como "Ragnar Warrior"

**força** - um valor entre 1 em 1000

**habilidade mental** - a habilidade mental de um guerreiro é sempre 0

**Poder de ataque** - o poder de ataque de um guerreiro deve ser calculado multiplicando por 10 o valor da força

**Esquiva** - habilidade com valor entre 0 e 50, e representa a chance do personagem desviar de um ataque

**resistência** - o valor da resistência deve ser um valor entre 0 e 90 e indica o percentual do dano sofrido que será absorvido

**vida máxima** - representa o total de vida do personagem (1 a 40000)

**vida atual** - quantidade atual de vida do personagem

# TAREFA 1 - GUERREIRO

## Uma solução

```
● ● ●  
1 export class Util {  
2   public static randomizar(minimo: number, maximo: number) {  
3     const valorSorteado =  
4       minimo + Math.random() * (maximo - minimo);  
5     const valorInteiro = Math.round(valorSorteado);  
6     return valorInteiro;  
7   }  
8 }
```

```
● ● ●  
1 import { Personagem } from "./Personagem";  
2 import { Util } from "./Util";  
3  
4 export class Guerreiro extends Personagem {  
5   constructor(nome: string) {  
6     super(  
7       nome + " Warrior",  
8       Util.randomizar(1, 1000),  
9       0,  
10      0,  
11      Util.randomizar(0, 50),  
12      Util.randomizar(0, 90),  
13      0,  
14      Util.randomizar(1, 40_000)  
15    );  
16    this._poderDeAtaque = this._forca * 10;  
17    this._vidaAtual = this._vidaMaxima;  
18  }  
19 }
```

# TAREFA 2 - GUERREIRO

?? minutos

## Ações:

**Atacar** - o movimento de atacar de um personagem deve receber por parâmetro um personagem a ser atacado. O ataque tem chance de falhar de acordo com a "esquiva" do atacado. A quantidade de vida retirada do atacado em caso de acerto, será o valor do poder de ataque reduzido em um percentual igual a resistência do atacado. Toda tentativa de ataque gera um contra-ataque automático do oponente.

**Contra-atacar** - Com a mesma mecânica do ataque, desenvolva o contra-ataque.

**Aprimorar Habilidade Principal** - Incrementa em 10% a força do personagem

**Regenerar vida** - Recupera 5% da vida do personagem

# TAREFA 2 - GUERREIRO

```
● ● ●  
1 public atacar(oponente: Personagem): void {  
2     console.log(`${this._nome} atacou ${oponente.nome}`);  
3     this.ataque(oponente);  
4     oponente.contraAtacar(this);  
5 }  
6  
7 public contraAtacar(oponente: Personagem): void {  
8     console.log(`${this._nome} contra-atacou ${oponente.nome}`);  
9  
10    this.ataque(oponente);  
11 }  
12  
13 public aprimorarHabilidadePrincipal(): void {  
14     this._forca *= this._forca * 1.1;  
15     this.atualizarPoderDeAtaque();  
16 }  
17 private atualizarPoderDeAtaque(): void {  
18     this._poderDeAtaque = this._forca * 10;  
19 }
```

```
● ● ●  
1 public regenerarVida(): void {  
2     this._vidaAtual += this._vidaAtual * 1.05;  
3     if (this._vidaAtual > this._vidaMaxima) {  
4         this.vidaAtual = this._vidaMaxima;  
5     }  
6 }  
7  
8 private ataque(oponente: Personagem): void {  
9     const acertou: boolean = Util.randomizar(0, 100) > oponente.esquiva;  
10    if (acertou) {  
11        const danoCausado: number =  
12            (1 - oponente.resistencia / 100) * this._poderDeAtaque;  
13        oponente.vidaAtual = oponente.vidaAtual - danoCausado;  
14  
15        const oponenteMorreu: boolean = oponente.vidaAtual ≤ 0;  
16        if (oponenteMorreu) {  
17            throw new Error(`${oponente.nome} foi derrotado.`);  
18        }  
19    } else {  
20        console.log(`${oponente.nome} esquivou o ataque de ${this._nome}`);  
21    }  
22 }
```

# TAREFA 3 - PRIEST

7 minutos

## Características:

**nome** - O nome de um padre deve possuir o sufixo "Priest". Ex.: Quando cadastrar um padre com o nome "Fabio de Melo", será registrado como "Fabio de Melo Priest"

**força** - a força de um padre é sempre 0

**habilidade mental** - a habilidade mental de um padre é sempre 0

**Poder de ataque** - o poder de ataque de padre é sempre 0

**Esquiva** - a esquiva de um padre é sempre 0

**resistência** - a resistência de um padre é sempre 0

**vida** - representa o total de vida do personagem (1 a 8000)

# TAREFA 3 - PRIEST

## Uma solução

```
● ● ●  
1 import { Personagem } from "./Personagem";  
2 import { Util } from "./Util";  
3  
4 export class Priest extends Personagem {  
5   constructor(nome: string) {  
6     super(nome + " Priest", 0, 0, 0, 0, 0, 0, Util.randomizar(1, 8_000));  
7     this._vidaAtual = this._vidaMaxima;  
8   }  
9 }
```

# TAREFA 4 - PRIEST

5 minutos

## Ações:

**Atacar** – O ataque de um padre tem 40% de chance de converter o oponente a seu favor, com isso a batalha encerra imediatamente e o padre é dado como vitorioso

**Contra-atacar** – Com a mesma mecânica do ataque, desenvolva o contra-ataque.

**Aprimorar Habilidade Principal** – O padre não treina habilidade. Na tentativa de treinar a habilidade de um padre um "Error" deve ser lançado, com a mensagem "Este personagem não pode executar esta ação".  
**Regenerar vida** – Recupera 10% da vida do personagem

# TAREFA 4 - PRIEST

## Uma solução

```
1 public atacar(oponente: Personagem): void {
2     console.log(`${this._nome} tentou converter ${oponente.nome}`);
3     this.ataque(oponente);
4     oponente.contraAtacar(this);
5 }
6
7 public contraAtacar(oponente: Personagem): void {
8     console.log(`${this._nome} tentou converter ${oponente.nome}`);
9     this.ataque(oponente);
10}
11
12 public aprimorarHabilidadePrincipal(): void {
13     throw new Error("Este personagem não pode executar esta ação");
14 }
```

```
1 public regenerarVida(): void {
2     this._vidaAtual += this._vidaMaxima * 0.1;
3     if (this._vidaAtual > this._vidaMaxima) {
4         this.vidaAtual = this._vidaMaxima;
5     }
6 }
7
8 private ataque(oponente: Personagem) {
9     const acertou: boolean = Util.randomizar(0, 100) < 40;
10    if (acertou) {
11        oponente.vidaAtual = 0;
12        throw new Error(`#${oponente.nome} foi convertido`);
13    }
14 }
```

## **TAREFA 4 - VALIDAÇÃO**

### **Missão:**

Criar um Main.ts que crie N personagens e façam eles batalharem uns com os outros, randomicamente.

**?? minutos**

# TAREFA 4 - TESTAGEM

Quem está atacando? Guerreiro? Mago? Priest? Personagem?  
Não sabemos. Aqui temos um comportamento **polimórfico!**



```
1 import prompt from "prompt-sync";
2 import { Guerreiro } from "./entities/Guerreiro";
3 import { Priest } from "./entities/Padre";
4 import { Personagem } from "./entities/Personagem";
5 import { Util } from "./helpers/Util";
6
7 const teclado = prompt();
8
9 let personagens: Personagem[] = [];
10 personagens.push(new Priest("Fábio de Melo"));
11 personagens.push(new Guerreiro("Ragnar"));
12 personagens.push(new Priest("Quemedo"));
13 personagens.push(new Guerreiro("Genghis Khan"));
14 personagens.push(new Guerreiro("Alexandre, o Grande"));
15
```

O método '**resumo**' foi criado em Personagem para apresentar nome e energia do personagem.

```
1 while (true) {
2   console.log(`===== Personagens vivos (${personagens.length}) =====`);
3   personagens.forEach((personagem) => console.log(personagem.resumo()));
4   if (personagens.length === 1) {
5     break;
6   }
7   console.log("=====\n");
8
9   teclado("Tecle ENTER para rodar o próximo round\n");
10 try {
11   const atacantePosicao = Util.randomizar(0, personagens.length - 1);
12   const atacadoPosicao = Util.randomizar(0, personagens.length - 1);
13   if (atacantePosicao != atacadoPosicao) {
14     const atacante = personagens[atacantePosicao];
15     const atacado = personagens[atacadoPosicao];
16     atacante.atacar(atacado);
17     console.log(atacante.resumo());
18     console.log(atacado.resumo());
19     console.log("\n");
20     console.log(".".repeat(20));
21   }
22 } catch (e) {
23   personagens = personagens.filter((personagem) => personagem.vidaAtual
24 > 0);
25   console.log((e as any).message);
26 }
27 console.log(`\n0 vencedor foi \x1b[31m ${personagens[0].nome}\x1b[0m`);
```

# TAREFA 4 - TESTAGEM

Tecle ENTER para rodar o próximo round

Quemedo Priest tentou converter Genghis Khan Warrior

Genghis Khan Warrior foi convertido

===== Personagens vivos (2) =====

Ragnar Warrior: 28635.0/28635

Quemedo Priest: 1933.0/4373



```
1 import prompt f
2 import { Guerre
3 import { Priest
4 import { Person
5 import { Util }
6
7 const teclado =
8
9 let personagens
10 personagens.push
11 personagens.push
12 personagens.push
13 personagens.push
14 personagens.push
```

O método 'resumo' foi criado em Personagem para apresentar nome e energia do personagem.

```
length}) == `);
sonagem.resumo()));
```

```
agens.length - 1);
agens.length - 1);
```

```
personagem.vidaAtual > 0);
```

Tecle ENTER para rodar o próximo round

===== Personagens vivos (2) =====

Ragnar Warrior: 28635.0/28635

Quemedo Priest: 1933.0/4373

Tecle ENTER para rodar o próximo round

Quemedo Priest tentou converter Ragnar Warrior

Ragnar Warrior contra-atacou Quemedo Priest

Quemedo Priest foi derrotado.

===== Personagens vivos (1) =====

Ragnar Warrior: 28635.0/28635

O vencedor foi Ragnar Warrior

BGS\_BGS\_01 - Aula 04

# **TAREFA 5 - MAGO**

**Tema de casa. :)**